APPENDIX

D

THE HEXADECIMAL NUMBER SYSTEM AND MEMORY ADDRESSING

Fundamental to understanding how computers work is understanding the number system and the coding system that computers use to store data and communicate with each other. Early attempts to invent an electronic computing device met with disappointing results as long as inventors tried to use our own decimal number system, with the digits 0–9. No electronic device could be invented that would dependably hold 10 different numeric values. Attempts during the 1940s involved using vacuum tubes that held varying quantities of electrical charge. A very small charge would be a 1, a little more would be a 2, and so on up to 9, the strongest charge. Measuring the charge to determine what number it represented just wasn't reliable, however, because the electrical charges fluctuated too much. Then John Atanasoff came up with the brilliant idea of not measuring the charge at all. He proposed using a coding system that expressed everything in terms of different sequences of only two numerals: one represented by the presence of a charge and one represented by the absence of a charge. The existing technology could handle that. The numbering system that can be supported by the expression of only two numerals is the base 2, sometimes called binary, numbering system, invented by Ada Lovelace many years before, using the two numerals 0 and 1. Under Atanasoff's design, all numbers and other characters would be converted to this binary number system, and all storage, comparisons, and arithmetic would be done using it. Even today, this is one of the basic principles of computers. Every character or number entered into a computer is first converted into a series of 0s and 1s. Many coding schemes and techniques have been invented to manipulate these 0s and 1s, called **bits** for **bi**nary dig**its**.

The most widespread binary coding scheme for microcomputers, which is recognized as the microcomputer standard, is called the ASCII (American Standard Code for Information Interchange) coding system. (Appendix C lists the binary code for the basic 127-character set.) In ASCII, each character is assigned an 8-bit code called a **byte**. Table D-1 lists the terms used in the discussion of how numbers are stored in computers. The byte has become the universal single unit of storage for data in computers everywhere.

Table D-1 Computer terminology

Term	Definition
Bit	A numeral in the binary number system: a 0 or a 1
Byte	8 bits
Kilobyte	1,024 bytes, which is 2^{10}, often rounded to 1,000 bytes
Megabyte	Either 1,024 kilobytes or 1,000 kilobytes, depending on what has come to be standard practice in different situations. For example, when calculating floppy disk capacities, 1 megabyte = 1,000 kilobytes; when calculating hard drive capacity, traditionally, 1 megabyte = 1,024 bytes.
Gigabyte	1,000 megabytes or 1,024 megabytes, depending on what has come to be standard practice in different situations
ASCII	American Standard Code for Information Interchange coding scheme used for microcomputers, which assigns a 7- 8-bit code to all characters and symbols. See Appendix C for more information.
Hex	Short for hexadecimal. A number system based on 16 values (called base 16), which is explained in detail below. Uses the sixteen numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Hex numbers are often followed by a lowercase h to indicate they are in hex (example: 78h).

Human beings are accustomed to the decimal number system, and working with the binary number system is quite tedious, mostly because of the sheer volume of bits that have to be managed. But computers use the binary number system, and the calculations to convert binary to decimal are relatively complex. Therefore, a compromise arose. Computers convert binary data into the hexadecimal number system (shortened to hex system) because it is much less complex for computers to convert binary numbers into hex numbers than into decimal numbers, and it is much easier for human beings to read hex numbers than to read binary numbers. This way, even though the actual processing and inner workings of computers use the binary system, they often display information using the hex system.

Learning to "Think Hex"

One skill a knowledgeable computer support person must have is the ability to read hex numbers and convert hex to decimal and decimal to hex. Once you understand one numbering system (decimal), you can understand any numbering system (including binary and hexadecimal), because they all operate on the same basic principle: place value. So we begin there.

Place Value

A key to understanding place value is to think of a number system as a method of grouping multiple small units together until there are enough of them to be packed into a single larger group, then grouping multiple larger groups together until there are enough of them to form an even larger group, and so on. In our (decimal) number system, once there are 10 units of any group, that group becomes a single unit of the next larger group. So, groups of 10 units are packed into groups of tens; groups of ten tens are packed into groups of hundreds; groups of 10 hundreds are packed into groups of thousands, and so forth.

An easy way to understand number systems is to think of the numbers as being packaged for shipping, into boxes, cartons, crates, truckloads, and so on. For the decimal numbering system, consider packing widgets (units) into boxes (tens) which are packed into cartons (100s) which are packed into crates (1000s), and so forth. The same analogy works for binary, decimal, and all other number systems.

Our friend Joe, in Figure D-1, is a widget packer in the shipping department of the ACE Widget Co. Joe can ship single widgets, or he can pack them in boxes, cartons, crates, and truckloads. He can fit three, and only three, widgets to a box; three, and only three, boxes into one carton ($3 \times 3 = 9$ widgets); three, and only three, cartons into one crate ($3 \times 9 = 27$ widgets); and three, and only three, crates into one truck ($3 \times 27 = 81$ widgets). He is not allowed to pack more widgets into boxes, cartons, crates, or truckloads than those specified. Neither is he allowed to send out a box, carton, crate, or truckload that is not completely filled.

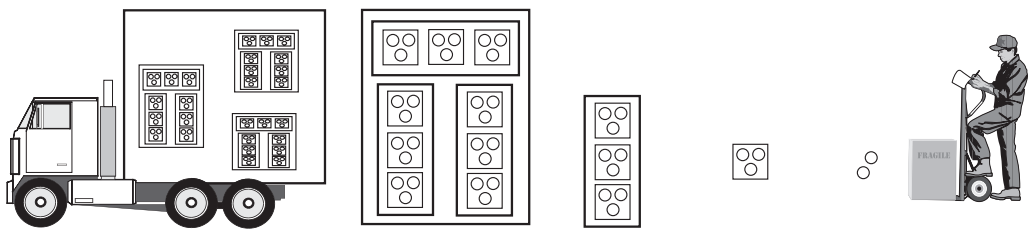

Figure D-1 Joe in the shipping department groups widgets in singles, boxes, cartons, crates, and truckloads—all in groups of three

Joe receives an order to ship out 197 widgets. How does he ship them? The answer is shown in Figure D-2. Joe sends out 197 widgets grouped into 2 truckloads ($2 \times 81 = 162$ widgets), 1 crate (27 widgets), no cartons, 2 boxes ($2 \times 3 = 6$ widgets), and 2 single widgets. We can write this grouping of widgets as 21022, where the "place values" from left to right are truckloads, crates, cartons, boxes, and units, which in this case are (in decimal) 81 widgets, 27 widgets, 9 widgets, 3 widgets, and single widgets. Notice that each "place value" in our widget-packing system is a multiple of 3, because the widgets are grouped into three before they are packed into boxes; the boxes are grouped into three before they are packed into cartons, and so on. By grouping the widgets into groups of 3s in this manner, we converted the decimal number (base 10) 197

into the ternary number (base 3) 21022. Joe's widget-packing method is a base three, or ternary, system. The numerals in the ternary number system are 0, 1, and 2. When you get to the next value after 2, instead of counting on up to 3, you move one place value to the left and begin again with 1 in that position, which represents 3. So, counting in base 3 goes like this: 0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101 and so on. This is the same as Joe's never shipping out 3 of any one group unless they are packed together into one larger group. For example, Joe wouldn't ship 3 individual boxes, he would ship one carton.

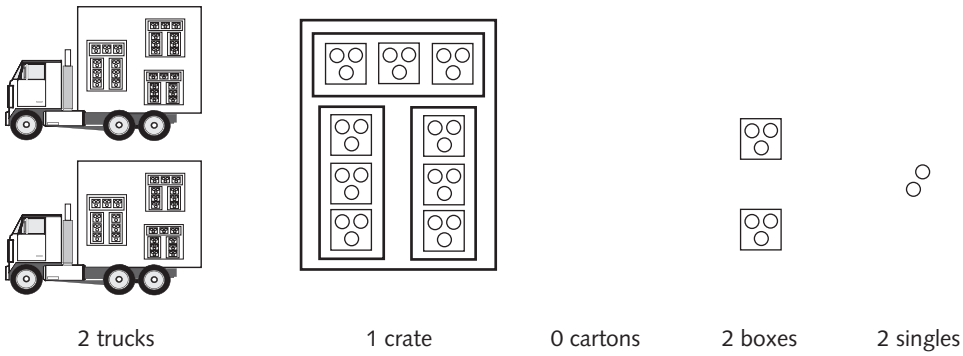

Figure D-2 Joe's shipment of 197 widgets: 2 trucks, 1 crate, 0 cartons, 2 boxes, and a group of 2 singles

You can easily apply the widget-packing analogy to another base. If Joe packed 7 (and only 7) units to a box, 7 (and only 7) boxes to a carton, 7 (and only 7) cartons to a crate, and 7 (and only 7) crates to a truck, then his packing system would be operating using base seven rules. If he used 10 instead of seven, he would be using base ten (decimal) rules. So, numbering systems differ by the different numbers of units they group together. In the hex number system, we group by 16. So, if Joe were shipping in groups of 16, as in Figure D-3, single widgets could be shipped out up to 15, but 16 widgets would make one box. Sixteen boxes would make one carton, which would contain 16×16, or 256, widgets. Sixteen cartons would make one crate, which would contain 16×256, or 4,096, widgets.

Suppose Joe receives an order for 197 widgets to be packed in groups of 16. He will not be able to fill a carton (256 widgets), so he ships out 12 boxes (16 widgets each) and 5 single widgets:

$12 \times 16 = 192$, and $192 + 5 = 197$

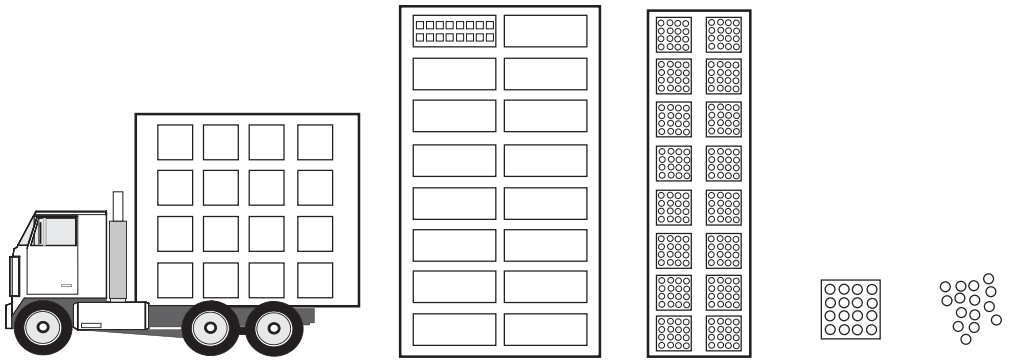

Figure D-3 Widgets displayed in truck loads, crates, cartons, boxes, and singles grouped in 16s

You approach an obstacle if you attempt to write the number in hex. How are you going to express 12 boxes and 5 singles? In hex, you need single numerals in hex to represent the numbers 10, 11, 12, 13, 14, and 15 in decimal. Hex uses letters for these numerals. A, B, C, D, E, and F are used for the numbers 10 through 15. Table D-2 shows values expressed in the decimal, hex, and binary numbering systems. In the second column in Table D-2, you are counting in the hex number system. For example, 12 is represented with a C. So you say that Joe packs C boxes and 5 singles. The hex number for decimal 197 is C5 (see Figure D-4).

Table D-2 Decimal, hex, and binary values

Decimal	Hex	Binary	Decimal	Hex	Binary	Decimal	Hex	Binary
0	0	0	14	E	1110	28	1C	11100
1	1	1	15	F	1111	29	1D	11101
2	2	10	16	10	10000	30	1E	11110
3	3	11	17	11	10001	31	1F	11111
4	4	100	18	12	10010	32	20	100000
5	5	101	19	13	10011	33	21	100001
6	6	110	20	14	10100	34	22	100010
7	7	111	21	15	10101	35	23	100011
8	8	1000	22	16	10110	36	24	100100
9	9	1001	23	17	10111	37	25	100101
10	A	1010	24	18	11000	38	26	100110
11	B	1011	25	19	11001	39	27	100111
12	C	1100	26	1A	11010	40	28	101000
13	D	1101	27	1B	11011			

For a little practice, calculate the hex values of the decimal values 14, 259, 75, and 1,024 and the decimal values of FFh and A11h.

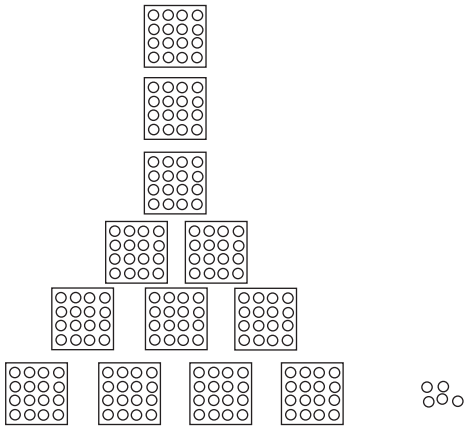

Figure D-4 Hex C5 represented as C boxes and 5 singles = 197 decimal

How Exponents Are Used to Express Place Value

If you are comfortable with using exponents, you know that writing numbers raised to a power is the same thing as multiplying that number times itself the power number of times. For example, $3^4 = 3 \times 3 \times 3 \times 3 = 81$. Using exponents in expressing numbers can also help us easily see place value, because the place value for each place is really the base number multiplied by itself a number of times, based on the place value position . For instance, look back at Figure D-1. A truckload is really $3 \times 3 \times 3 \times 3$, or 81, units, which can be written as 3^4. A crate is really $3 \times 3 \times 3$, or 27, units. The numbers in Figure D-1 can therefore be written like this:

Truckload = 3^4 Crate = 3^3 Carton = 3^2 Box = 3^1 Single = 3^0

(Any number raised to the 0 power equals 1.) Therefore, we can express the numbers in Figure D-2 as multiples of truckloads, crates, cartons, boxes, and singles like this:

	Truckloads	Crates	Cartons	Boxes	Singles
21022 (base 3)	2×3^4	1×3^3	0×3^2	2×3^1	2×3^0
Decimal equivalent	162	27	0	6	2

When we sum up the numbers in the last row above, we get 197. We just converted a base 3 number (21022) to a base 10 number (197).

Binary Number System

It was stated earlier that it is easier for computers to convert from binary to hex or from hex to binary than to convert between binary and decimal. Let's see just how easy. Recall that the binary number system only has two numerals, or bits: 0 and 1. If our friend Joe in shipping operated a "binary" shipping system, he would pack like this: 2 widgets in a box,

2 boxes in one carton (4 widgets), two cartons in one crate (8 widgets), and two crates in one truckload (16 widgets). In Figure D-5, Joe is asked to pack 13 widgets. He packs 1 crate (8 widgets), 1 carton (4 widgets), no boxes, and 1 single. The number 13 in binary is 1101:

$$(1 \times 2^3)+(1 \times 2^2)+(0 \times 2^1)+(1 \times 2^0)= 8 + 4 + 0 + 1 = 13$$

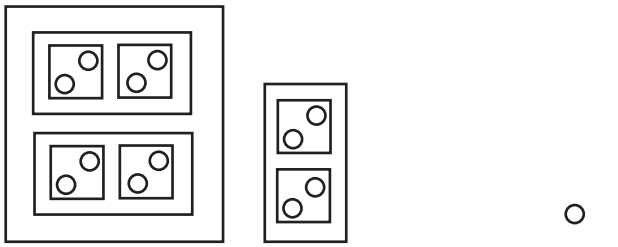

Figure D-5 Binary 1101 = 13 displayed as crates, cartons, boxes, and singles

Now let's see how to convert binary to hex and back again. The largest 4-bit number in binary is 1111. This number in decimal and hex is:

```
binary 1111 = 1 group of 8 = 8
              1 group of 4 = 4
              1 group of 2 = 2
                  1 single = 1
                             ---
                    TOTAL = 15 (decimal)
Therefore, 1111 (binary) = 15 (decimal) = F (hex)
```

This last calculation is very important when working with computers: F is the largest numeral in the hex number system and it only takes 4 bits to write this largest hex numeral: F (hex) = 1111 (binary). So, every hex numeral (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F) can be converted into a 4-bit binary number. Look back at the first 16 entries in Table D-2 for these binary values. Add leading zeroes to the binary numbers as necessary.

When converting from hex to binary, take each hex numeral and convert it to a 4-bit binary number and string all the 4-bit groups together. Fortunately, when working with computers, you will almost never be working with more than 2 hex numerals at a time. Here are some examples:

1. To convert hex F8 to binary, do the following: F = 1111, and 8 = 1000. Therefore, F8 = 11111000 (usually written 1111 1000).
2. To convert hex 9A to binary, do the following: 9 = 1001, and A = 1010. Therefore, 9A = 1001 1010.

Now try converting from binary to hex:

1. To convert binary 101110 to hex, first group the bits in groups of 4, starting at the right and moving left, adding leading zeros as necessary: 0010 1110.

2. Then convert each group of 4 bits in binary to a single hex numeral: 0010 = 2, and 1110 = E. The hex number is 2E.

Writing Conventions

Sometimes when you are dealing with hex, binary, and decimal numbers, it is not always clear which number system is being used. If you see a letter in the number, you know the number is a hex number. Binary numbers are usually written in groups of four bits. This book follows the convention of placing a lowercase h after a hex number, like this: 2Eh.

Memory Addressing

Computers often display memory addresses in the hex number system. You must either "think in hex" or convert to decimal. It's really easier, with a little practice, to think in hex. Here's the way it works:

Memory addresses are displayed as two hex numbers. An example is C800:5

The part to the left of the colon (C800) is called the **segment address**, and the part to the right of the colon (5) is called the **offset**. The offset value can have as many as four hex digits. The actual memory address is calculated by adding a zero to the right of the segment address and adding the offset value, like this: C800:5 = C8000 + 5 = C8005

The first 640K of memory is called conventional memory. Look at how that memory is addressed, first in decimal and then in hex (assuming 1 kilobyte = 1,024 bytes):

640K = 640 × 1,024 = 655,360

There are 655,360 memory addresses in conventional memory, where each memory address can hold 1 byte, or 8 bits, of either data or program instructions. The decimal value 655,360 converted to hex is A0000 (10×16^4). So, conventional memory addresses begin with 00000h and end with A0000h minus 1h or 9FFFFh. Written in segment-and-offset form, conventional memory addresses range from 0000:0 to 9FFF:F.

Recall that upper memory is defined as the memory addresses from 640K to 1,024K. The next address after 9FFF:F is the first address of upper memory, which is A0000, and the last address is FFFFF. Written in segment-and-offset terms, upper memory addresses range from A000:0 to FFFF:F.

Here is one way to organize the conversion of a large hex value such as FFFFF to decimal (remember F in hex equals 15 in decimal).

FFFFF converted to decimal:

$15 \times 16^0 = 15 \times 1 =$ 15
$15 \times 16^1 = 15 \times 16 =$ 240
$15 \times 16^2 = 15 \times 256 =$ 3,840
$15 \times 16^3 = 15 \times 4096 =$ 61,440
$15 \times 16^4 = 15 \times 65,536 =$ 983,040

TOTAL = 1,048,575

Remember that FFFFF is the last memory address in upper memory. The very next memory address is the first address of extended memory, which is defined as memory above 1 MB. If you add 1 to the number above, you get 1,048,576, which is equal to 1024 × 1024, which is the definition of 1 megabyte.

Displaying Memory with DOS DEBUG

In Figure D-6 you see the results of the beginning of upper memory displayed. The DOS DEBUG command displays the contents of memory. Memory addresses are displayed in hex segment-and-offset values. To enter DEBUG, type the following command at the C prompt and press **Enter**:

C:\> DEBUG

Type the following dump command to display the beginning of upper memory (the hyphen in the command is the DEBUG command prompt) and press **Enter**:

-d A000:0

Memory is displayed showing 16 bytes on each line. The A area of memory (the beginning of upper memory) is not used unless the computer is using a monochrome monitor or this area is being used as an upper memory block. In Figure D-6, the area contains nothing but continuous 1s in binary or Fs in hex. The ASCII interpretation is on the right side. To view the next group of memory addresses, you can type **d** at the hyphen and press **Enter**. DEBUG displays the next 128 addresses.

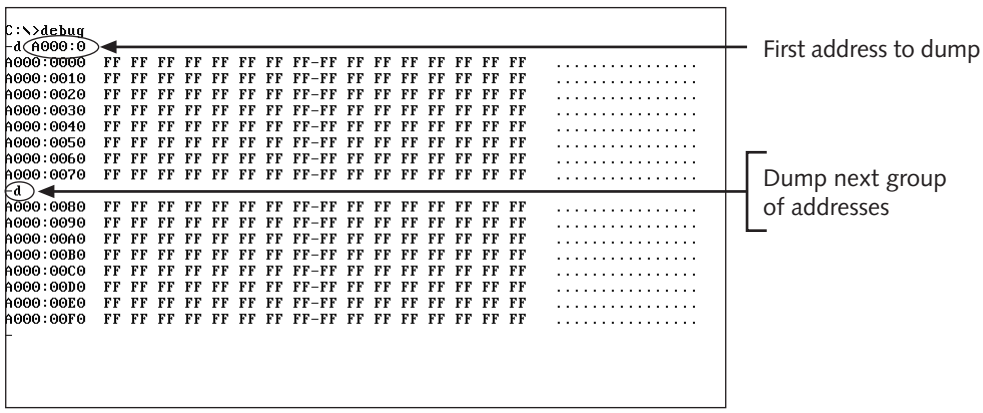

Figure D-6 Memory dump: `-d A000:0`

The A and B ranges of upper memory addresses (upper memory addresses that begin with A or B when written in hex) are used for monochrome monitors. The C range contains the video BIOS for a color monitor. Figure D-7 shows the dump of the beginning of the C range.

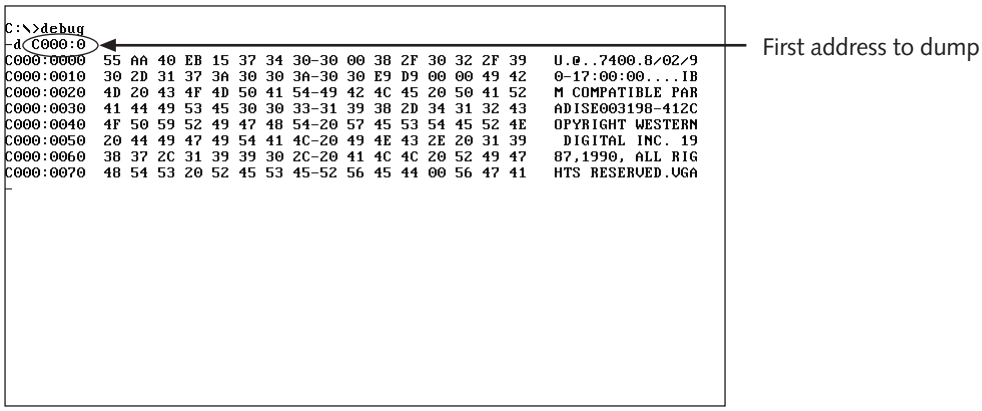

Figure D-7 Memory dump: `-d C000:0`

There is more than one way—in fact there are many ways—to identify the same segment-and-offset value. Try these commands to display the same upper memory addresses:

-d C000:0

-d BFF1:00F0

–d BFFF:0010

–d BEEE:1120

In summary, reading and understanding binary and hex numbers are essential skills for managing computers. All data is stored in binary in a computer and is often displayed in hex. Memory addresses are often displayed in hex segment-and-offset terms. An address in memory can be written in a variety of segment-and-offset values. The actual memory address is calculated by placing one zero to the right side of the segment address and adding the resulting value to the offset value. To exit DEBUG, type **q** for quit at the hyphen prompt. For more information about DEBUG, see Appendix F.